The CMS Databases Trail

Thoughts from Fermilab
(for discussion at CMS DB WS)

Lee Lueking

January 28, 2005

Overview

- The Equipment Management DB and Beyond (A generalized design).
- Calibration and Slow Controls DBs (Conditions).
- APIs and interfaces.
- General plan: schemas, schema owners, roles, naming conventions.

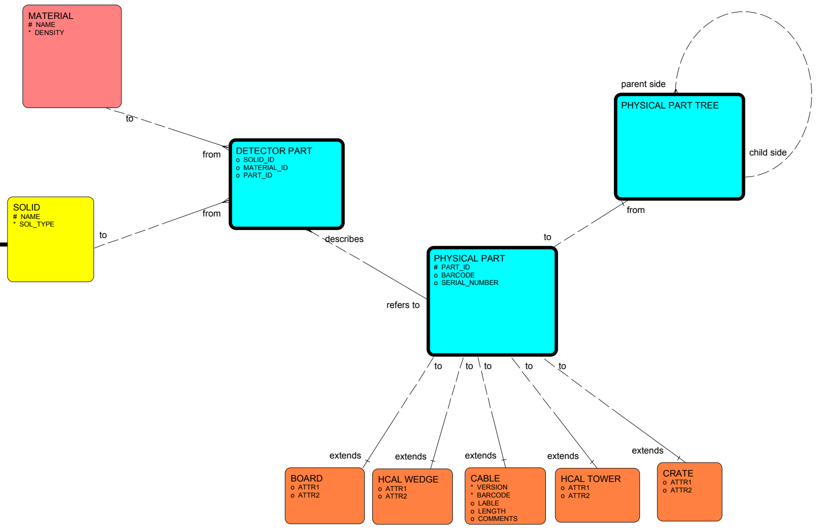
Thanks to Gennadiy Lukhanin, Yuyi Guo, and Sergey Kosyakov for their contributions.

Equipment Management DB

- The Hardware “parts” going into the CMS detector are currently being cataloged in the Equipment Management Database (EMDB).
- A convenient browsing tool is provided through the Rack Wizard.
- Work has been done to extend this to include the relationships between the parts in the geometrical, mechanical, and electrical contexts.

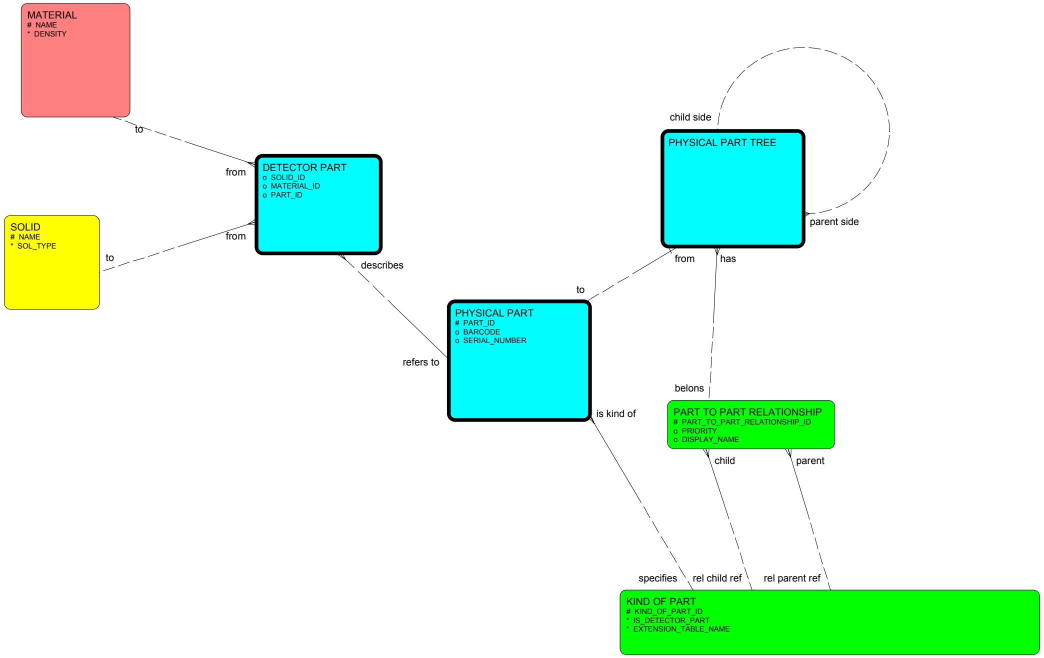
Equipment Management DB

- Each sub-detector has a set of “parts”. Each part has a part specific table; For example for PIXEL, crate, rack, module, Read Out Chip (ROC), HDI, Blade, Disk are parts.
- All part tables are foreign keyed to a “physical_part” table. This is a super-type level, and is the “parent” to all part tables.
- A “kind_of_part” is a catalog of the part table names.
- Valid relationships are defined in the “part_to_part_relationship” table. (They can also be defined explicitly via FK between specific part tables.)



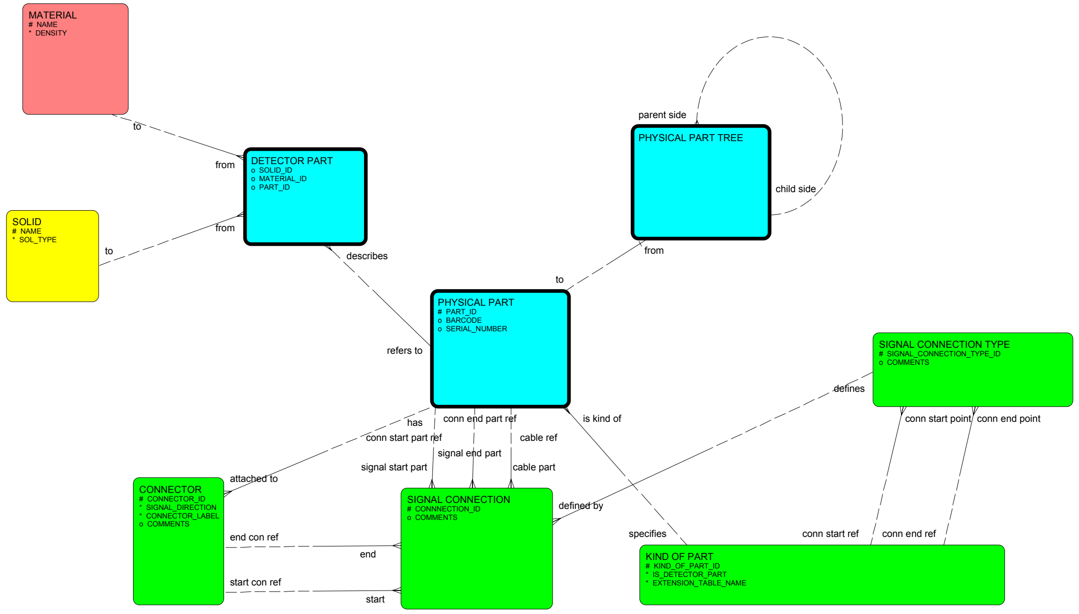
EMDB Augmented Schema (1 of 3)

- Physical part relationships, construction or geometric hierarchy, are constrained through the "part_to_part_relationship" table.



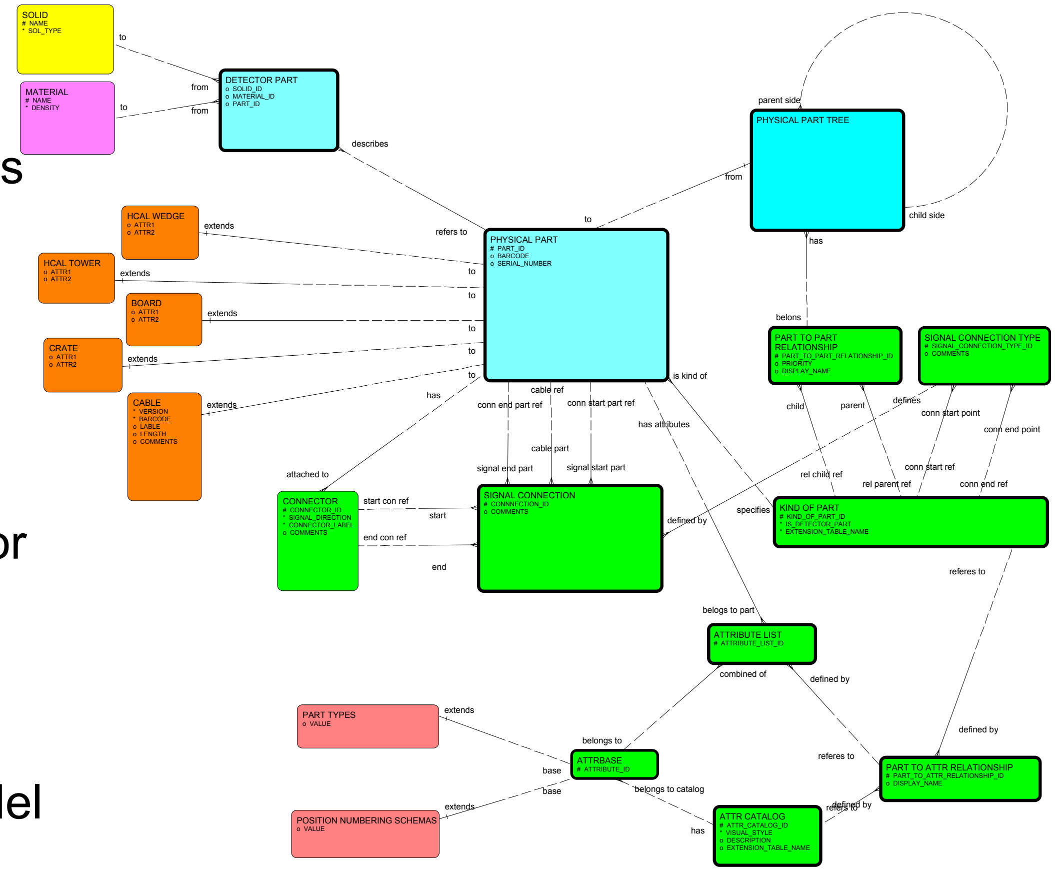
EMDB Augmented Schema (2 of 3)

- signal connections, are represented in the “signal_connections” and “signal_connections_type” tables.
- Additional relationships could be added if needed.



Augmented Schema (3 of 3)

- Some design-time constraints are replaced with *data driven* constraints.
- Allows to establish relationships between detector parts at the part-type level.
- Each detector component type (*part type*) has a history table being populated by databased triggers. History can be used for later analysis.
- This open design approach ensures the flexibility to incorporate additional data model features.



Design Benefits

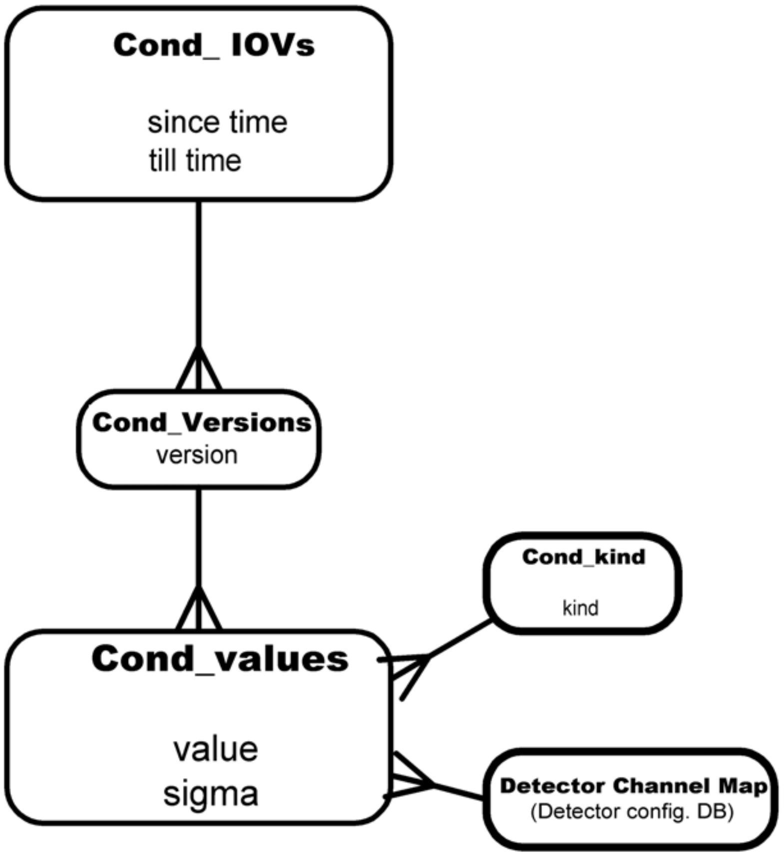
- Each detector part is uniquely identifiable in the system.
- All detector part relationships are defined at the top level instead of navigating through component hierarchy.
- Simple schema changes are needed to add a new detector part (*one table and one foreign key to the "physical parts" table*).
- Part to part relationships can be defined at any time by the authorized database user.
- "out of the box" support by most O/R (Object Relations) mapping tools e.g. TOP LINK , OJB , ADO.
- Simplifies and generalizes object-to-object navigation in APIs e.g. GUI and data loading tools.

Conditions DB

- Need to “capture” calibration (pedestals, gains, timing offsets, …), or monitoring information (HV, LV, currents, temperatures, …).
- Values captured are valid for certain time period called Interval of Validity (IOV).
- In the case of calibration, values are subject to change (re-calibration or re-calculation) and must have ability to “version” them.
- A tagging mechanism is required to readily identify certain data which can be used together, for ORCA for example.

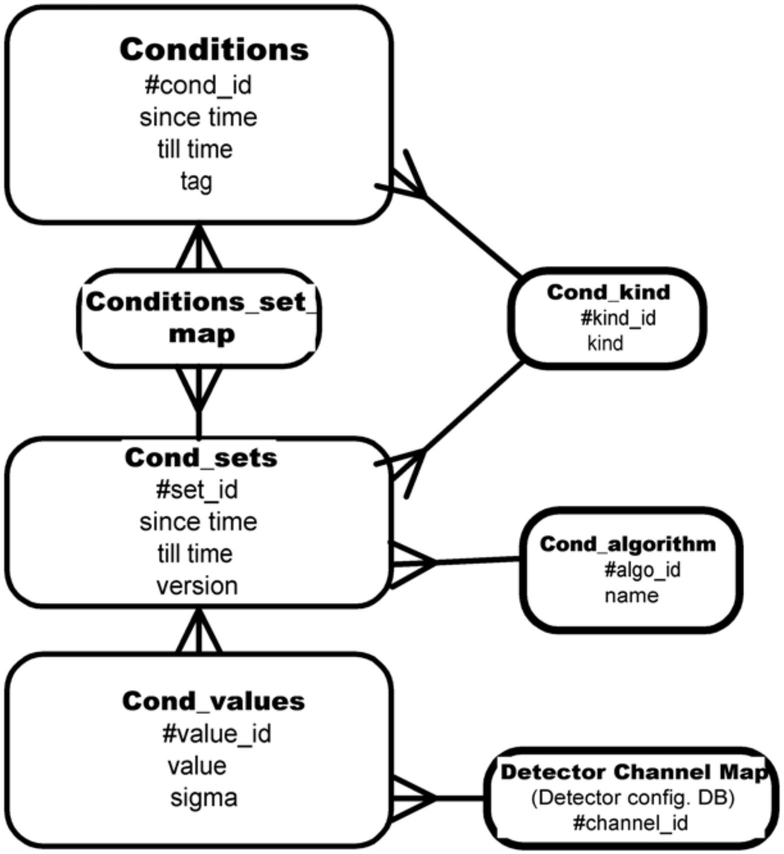
Conditions DB (1 of 3)

- The simplest case for DCS or calibration. All channels for a particular sub-detector are calibrated together.
- Each IOV can have one or more sets of conditions values which are mapped to the detector through a channel map.
- A cond_kind table distinguishes between pedestals, gains, time offsets, temperatures,…

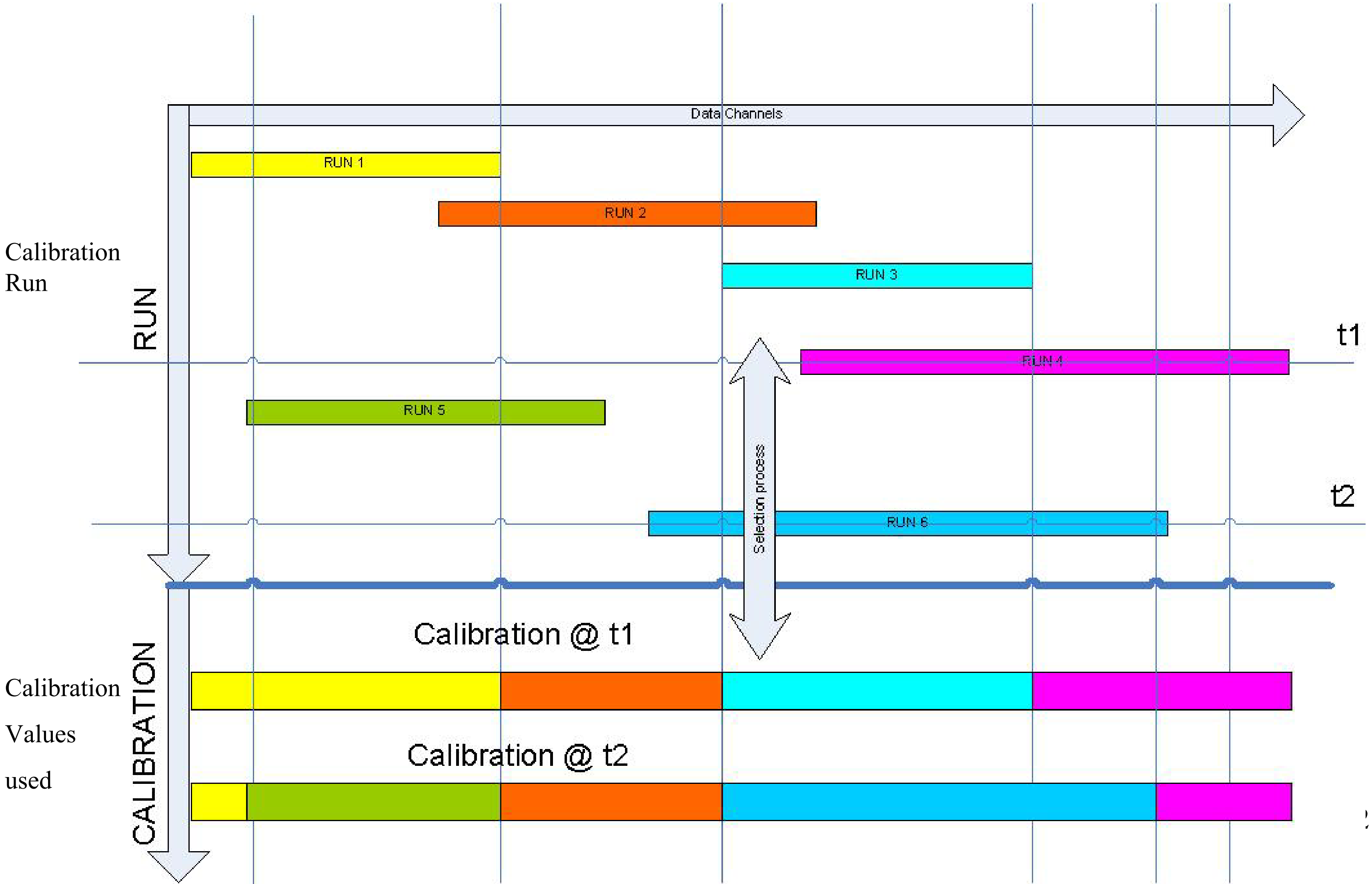


Conditions DB (2 of 3)

- In HCAL the ability to generate a "cond_set" of calibration values is needed so portions of the HCAL detector can be calibrated at different times.
- A top level table called "conditions" is added, and a mapping table which provides access to all information needed for a complete HCAL calibration.
- A "cond_kind" and "cond_algorithm" fully specify what the calibration is for and what was used to generate it.
- The "kind" in "cond_kind" is naively pedestal or gain. The reality is this needs to be more complex than this.
- The "name" and "version" in "cond_algorithm" are make up the version information.

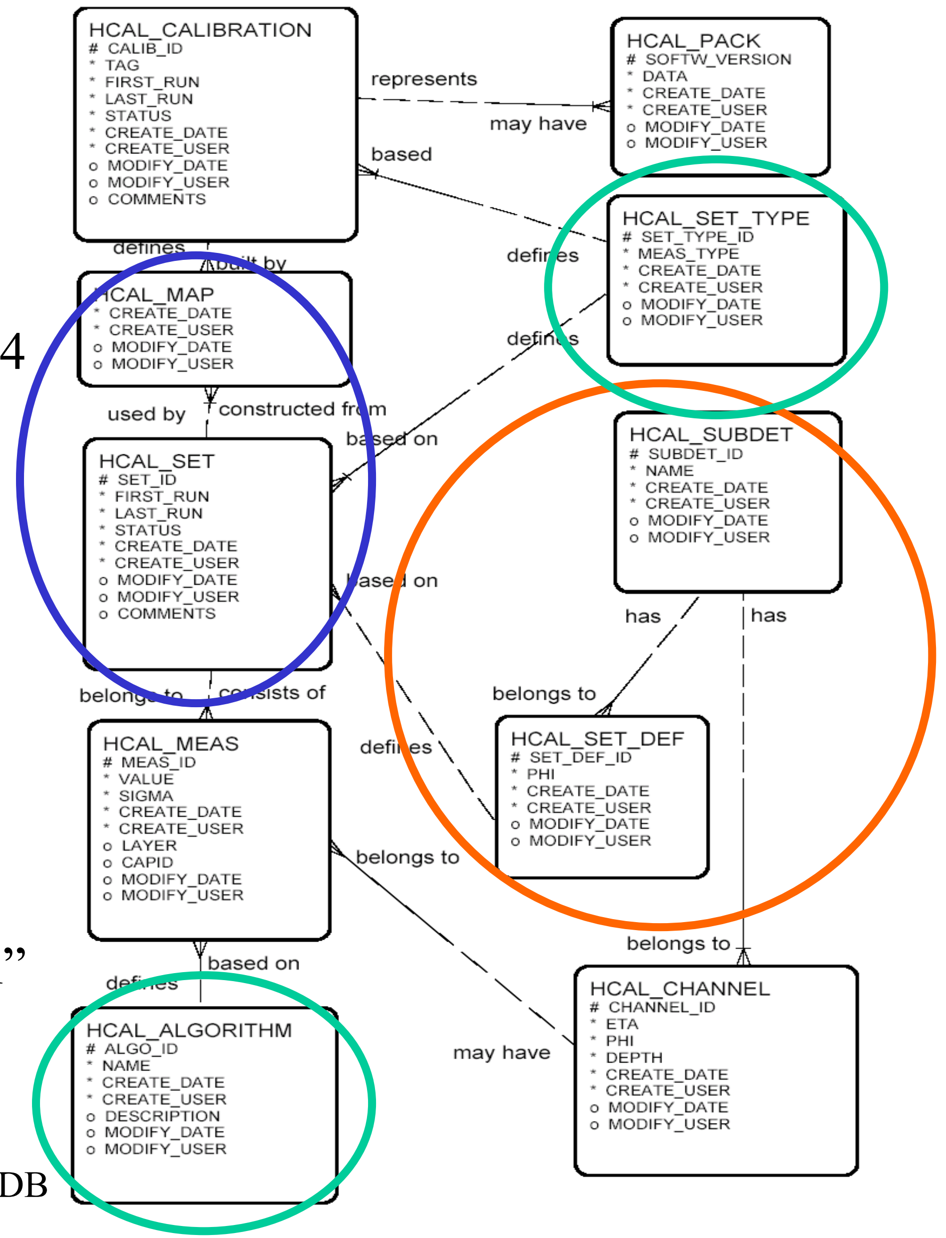


HCAL Calibration Approach



Calibration DB

- This is the existing HCAL calibration database used for 2004 testbeam calib data.
- Note the tables used to define subsets of the HCAL detector.
- Other sub-detectors may be interested in a similar way of defining and mapping subsets.
- The hcal_algorithm and hcal_set_type tables need to be understood better and connected somehow to provide the "version" and "revision" tracking.



ETL (Extraction/Transformation/Loading)

- Some examples of tools that exist:
  - Loading conditions data, for example tools to load pedestals and gains. (Existing examples for HCAL).
  - Loading test information into database, for example construction test data of plaquettes for PIXELS.
  - Loading parts into equipment management DB and the relationships between them. (Existing example for HCAL)
  - PL/SQL scripts to load HV monitoring data. (Existing examples for HCAL).
  - ORCA API to access HCAL data through Frontier (next slide).
  - GUI based tools for browsing EMDB. (Rack Wizard)
- We need a list of what is existing and what is needed by each sub-system.

Access via Frontier

- ORCA/Frontier read-only interface exists for HCAL
  - Retrieves pedestals and gains.
  - Used to access calibration info form TB conditions DB.
  - Can easily be extended for other sub-detectors.
- Writing via Frontier is under investigation.
  - Provides straightforward client API to load the DB
  - Issues of authentication and authorization are essential for writing.

The FrontierCalib.h Interface

It exists and works. It uses the standard COBRA/CARF CondDB interface.

```
namespace frontier_calib{

struct CalibData

 {std::vector<int> eta; std::vector<double> phi; std::vector<int> depth;
std::vector<double> value;  std::vector<double> sigma;};

struct GainErrorAllByTagRun : public CalibData {};

struct PedestalErrorAllByTagRun : public CalibData {};

struct RunInterval

 { long long calib_id;  long long set_type_id; long long run_first; long long run_last;  };

  template<class T> void get(T *vd,RunInterval &ri,const std::string &tag,long long
runnumber);

 }; // End of frontier_calib namespace
```

General Organization

- Each sub-system will have, minimally, the following table spaces
  - One data,
  - One index,
  - One blob
- Schema owner Responsibility
  - Each sub-system will own the tables in their schemas.
  - Developers for the sub-system can add and modify tables and relationships for the schemas they own.
- Database Administrator (DBA) responsibility.
  - Create DB accounts.
  - Backup and recovery.
  - Review schemas
  - Schema deployment and changes for production instance.
  - Daily database maintenance.

General Database Roles

- Roles for data access and modification:
  - Write (insert, update):
    - Online: Loading values for calibration and alignment, monitoring and slow controls. (DCS)
    - Offline: Loading values for calibration and alignment.
  - Read-only (select):
    - Online: Accessing detector and front end electronics configuration. Accessing calibration and alignment for HLT.
    - Offline: Accessing calibration and alignment for ORCA. Accessing beam and detector configuration for analysis.
  - Admin (select, insert, delete, update):
    - Online and Offline: Experts make critical changes as needed.

Naming Conventions

- Tables in each sub-detector’s shema can have the same name as other sub-detector schemas. These will be resolved in one of the following ways:
  - Public synonym, e.g. PIXEL_TABLE, HCAL_TABLE (32 characters max).
  - Use schema owner, e.g. SchemaOwner. Table
- DB vendor independent: Table names, column names, and data types should be compatible with MySQL and PostgreSQL standards. (See for example Dennis box’ list for some rules: http://home.fnal.gov/~dbox/SQL_API_Portability.html)

Summary

- By adding functionality to the existing EMDB in a modular way, we can also manage construction and configuration information.
- A conditions DB design that includes IOV management, as well as accommodates subsets of the sub-detector, and algorithm versioning might prove useful to other sub-detector groups. It can still use some improvement.
- We hope that the APIs for the augmented EMDB and the Conditions DB can be designed and built soon, and used universally for online and offline applications.
- Following some general organizational guidelines, and assigning the right roles for each application will help us to more easily build and operate the system.